# Multifaceted Resource Management for Dealing with Heterogeneous Workloads in Virtualized Data Centers

Íñigo Goiri, J. Oriol Fitó, Ferran Julià, Ramón Nou, Josep Ll. Berral, Jordi Guitart and Jordi Torres
*Universitat Politècnica de Catalunya and Barcelona Supercomputing Center*
*Jordi Girona 31, 08034 Barcelona, Spain*
{*igoiri,fito,fjulia,rnou,berral,jguitart,torres*}*@ac.upc.edu*

***Abstract*—As long as virtualization has been introduced in data centers, it has been opening new chances for resource management. Now, it is not just used as a tool for consolidating underused nodes and save power, it also allows new solutions to well-known challenges, such as fault tolerance or heterogeneity management. Virtualization helps to encapsulate Web-based applications or HPC jobs in virtual machines and see them as a single entity which can be managed in an easier way.**

**This paper proposes a new scheduling policy to model and manage a virtualized data center which mainly focuses on the allocation of VMs in data center nodes according to multiple facets while optimizing the provider's profit. In particular, it considers energy efficiency, virtualization overheads, fault tolerance, and SLA violation penalties, while adding the ability to outsource resources to external providers.**

**Using our approach, a data center can improve the provider's benefit by 15% and get a power reduction while solving well-known challenges, such as fault tolerance and outsourcing, in a better a more intuitive way that typical approaches do.**

***Keywords*-Fault tolerance; Provider's profit; Resource management; Virtualized data center**

## I. Introduction

Data centers have undergone a metamorphosis during the last years because of virtualization. This technology has been initially used to consolidate different tasks in the same node to avoid its underutilization. Thanks to this consolidation, a big power efficiency has been reached [1], [2]. Nevertheless, this is not the only advantage of virtualizing resources since this technology brings new capabilities, such as migration or checkpointing, which open many paths in IT resource management.

As a result of this encapsulation in virtual machines (VMs), heterogeneous tasks such as HPC jobs and Web-based applications can be considered as a single entity that is easier to manage. In addition, from a business point of view, a provider can offer these VMs in a pay-as-you-go manner. This is the basis of the Infrastructure as a Service (IaaS) paradigm, which relies on virtualized data centers. However, these data centers cannot be managed in the same way as the traditional ones were as they are now confronted to a number of new challenges. First, they must be able to deal with the new virtualization capabilities efficiently. Second, they receive economical and social pressure to reduce their energy consumption. Third, they are contractually bound to provide high availability and performance to their users. In the final analysis, all these challenges converge in that these data centers must be economically profitable.

For this reason, we propose a novel way of managing a virtualized data center, which mainly focuses on the allocation of VMs in data center nodes according to multiple facets while optimizing the provider's profit. The final profit for the provider is used as a reference value for all the allocation decisions. We consider the revenue obtained by executing a job or hosting a given Web-based application, and we assign different economic costs to the multiple facets of any VM placement, namely its power consumption, its fault tolerance, the incurred SLA violation penalties, etc.

It is important to note that some of these facets have been brought to a new dimension with the appearance of Cloud computing and virtualization, which has influence on how they have to be accomplished. One example is scheduling with energy efficiency in mind, which is not longer limited to statically consolidate and power off some unused nodes [3]. Nowadays, thanks to VM migration, a long running job can be moved dynamically from an underused node to another mostly full [4]. Similarly, the enhanced encapsulation that comes with virtualization permits checkpointing any application in a transparent way, which improves fault tolerance in the data center. Nonetheless, application nature needs to be considered at this stage since there is no need to checkpoint stateless applications. Moreover, a data center in the Cloud is not limited to its local resources, since it can outsource VMs to other federated IaaS providers when a peak load occurs and it cannot be solved with the local resources.

Nevertheless, apart from enabling these new approaches to well-known problems, virtualization also incurs some overheads, such as VM creation and instantiation (which can take minutes), VM migration, VM checkpointing, or the extra overhead added by the virtualization hypervisor. Of course, a good management policy should also consider them when taking decisions.

Keeping in mind all these features and problems, this paper proposes a holistic VM scheduling policy among nodes of a data center, including outsourcing to other providers. Those presented issues are analyzed in this policy in an

978-1-4244-9349-4/10/$26.00 © 2010 IEEE

unified way, modeling them as costs or revenues depending on their nature. Then, the policy tries to get the best VM allocation aiming to maximize the provider's benefit.

The remainder of the paper is organized as follows: Section II discusses some related work; the scheduling policy is presented in Section III; Section IV explains the actuators used in order to perform the management policy; Section V describes the experimental environment and Section VI evaluates the proposed approach; and finally, Section VII presents conclusions and future work.

## II. Related work

Maximizing providers' benefits is a hot research trend because of virtualization. In the last years, some works like [5] have studied the consolidation advantages of this technology, while others like [1], [2] have widely explored its advantages from a power efficiency point of view. Nevertheless, power management in cluster-based systems have been already addressed in classical IT resource management area. There are several works proposing energy management for servers that focus on applying energy optimization techniques in multiprocessor environments, such as [6]. In particular, [7] states that new power saving policies, such as Dynamic Voltage/Frequency Scaling (DVFS), or turning off idle servers can increase hardware problems as well as the problem to meet SLAs in this reduced environment. We propose, besides others, adding smarter scheduling policies to dynamically turn off idle machines to reduce the overall consumption. In addition, we rely in the underlying local node DVFS capabilities which automatically changes the frequency according to the load.

Economical approaches are also used for managing shared server resources in e.g. [8], where authors use a greedy resource allocation algorithm that allows distributing a web workload among different servers assigned to each service. This technique demonstrates to reduce server energy usage by 29% or more for a typical Web workload. [9] proposes a hybrid data center architecture that mixes low power systems and high performance ones. Moreover, [10] proposes and integrated management of application performance, dynamic workload migration and consolidation, power and cooling in data centers. Actually, the authors present a prototype and demonstrate that their integrated solution can reduce energy consumption of servers by 35% and cooling by 15%.

The use of virtualization for consolidation is presented in [2], which proposes a dynamic configuration approach for power optimization in virtualized server clusters and outlines an algorithm to dynamically manage it. This approach is also power efficient centered and takes into account the cost of turning on/off the servers. However, it can lead to a too slow decision process for an on-line scheduler like the one we are interested in. In addition, this work is highly centered in HPC jobs, while our proposal is based in both HPC jobs and Web-based services.

We propose the use of VM for executing those aforementioned heterogeneous applications taking into account virtualization overheads. Following the same idea, [4] aims to reduce virtualized data center power consumption by supporting VM migration and VM placement optimization while reducing the human intervention, but they do not provide any evaluation of their solution.

Furthermore, we state a methodology that tackles different challenges like fault tolerance and resources outsourcing. The first one has been addressed in [11] where they use virtualization and design a reconfigurable distributed virtual machine (RDVM) infrastructure. Despite it is focused on failure management, they use a similar node selection approach taking into account nodes reliability and tasks deadlines. Nevertheless, this approach is not focused on the aggregation of other costs such as virtualization overheads.

Distributing load of a provider among different data centers has been considered in [12]. Particularly, the authors propose the usage of different data centers according to its geographical distribution and power consumption. Moreover, our previous work [13] presents an approach which characterizes a federation of Clouds in order to enhance the providers' profit. However, both of these works only consider economical factors and the possibility to shutdown unused nodes.

In conclusion, note that all these previous research contributions only take into account individual factors in terms of managing a data center. On the contrary, we contribute here with an integrated and complete solution for managing virtualized data centers. Actually, we address all the emerging challenges in this kind of environments, such as virtualization overheads, reliability of nodes, the outsourcing of resources to third-party IaaS providers, and an accurate economical model concerning the operation and execution of a given data center.

## III. Scheduling algorithm

The proposed approach focus on maximizing the provider's benefit and it mainly consists in taking profit of the different capabilities a virtualized environment offers. Following this idea, we have modeled a virtualized data center based on the revenues and costs associated with each process, which includes typical ones like job execution, turn on or off nodes, and outsourcing; and other new features brought by virtualization such as customized environment creation, migration, and checkpointing.

Taking into account these features and the target of the provider (maximizing the overall benefit), the scheduling decides the best VM location at each moment based on the provider's benefit $B$, which is formally defined as the revenue minus the costs $B = R - \sum C_i$. Next subsections present each one of the costs and revenues associated with the different processes used and how they are taken into account by the policy.

Each VM allocation has a economic score, which results from the sum of all the individual revenues and costs (conceptually negative) associated with the possible execution of a VM in a host. There are different kinds of costs to be considered (boolean, time, power, and economics) and all these units needs to be merged in a common unit. We will use monetary units as is the more intuitive and understandable by both clients and providers.

### A. Requirements

First of all, scheduling must check if a host $h$ is able to hold a *VM*. This is performed by evaluating VM requirements $Req(vm)$, which include the hardware (e.g. the required system architecture, type and number of CPUs...), and the software (it has the libraries to execute an application, it uses a hypervisor, e.g. Xen, KVM, ...). In case the host is not available to execute that VM, the cost is set to infinity. Hence, it will act as a conditional statement which will avoid the execution of that VM into that host.

In addition to the static requirements, it checks if the host will have enough resources to execute all the VMs after adding this new one by checking the host occupation after allocating the new VM ($O(h, vm)$). The unfeasible situations are discarded considering it as a boolean function: renting available resources costs zero and infinity whether the resources in the tentative hosts are nonexistent or unavailable. From this description, $C_{req}(h, vm)$ is derived.

$$
\begin{aligned}
O(h, vm) &= \text{occupation of } h \text{ allocating } vm \\
C_{req}(h, vm) &= \begin{cases} \infty & \text{if } h \text{ cannot fulfill } Req(vm) \\ \infty & O(h, vm) > 1 \\ 0.0 & \text{otherwise} \end{cases}
\end{aligned}
$$

### B. VM Execution

The costs and revenues related with the execution of a task needs to be considered. We have the revenue obtained for executing a given VM according to its execution time ($R(vm)$) and we also have the costs associated with the execution of a VM (variable, $C_{var}(h, vm)$) and the costs of maintaining the infrastructure (fixed, $C_{fix}(h)$).

*1) Variable costs:* Those regarding the execution of a VM in a host, $C_{var}(h, vm)$, including power and external renting costs. The power consumption of a given host is directly proportional to its utilization, we calculate the power consumption for each VM in a given host ($C_{pwr}(h, vm)$), which is measured in KWh and can be easily converted according with the electricity pricing.

$$
\begin{aligned}
Pwr(h, o) &= \text{power consumption of } h \text{ at occupation } o \\
O(h, vm) &= \text{occupation of } h \text{ where } vm \text{ going to run} \\
Pwr(h, vm, o) &= \frac{Req(vm)}{\sum_{vm_i}^{h} Req(vm_i)} \cdot Pwr(h, O(h, vm)) \\
C_{pwr}(h, vm) &= Pwr(h, vm, o) \cdot Price(KWh)
\end{aligned}
$$

We may also have other variable costs such as those related with renting resources in other providers ($C_{rent}(h, vm)$). Merging those costs we get the total variable costs.

$$
C_{var}(h, vm) = \begin{cases} C_{pwr}(h, vm) & \text{if } h \text{ is a local host} \\ C_{rent}(h, vm) & \text{if } h \text{ is an external provider} \end{cases}
$$

*2) Fixed costs:* Those regarding host maintenance, $C_{fix}(h)$, including its amortization, or the space required to deploy it.

### C. Virtualization operation

One of the strengths of the proposal is its capability to deal with virtualization overheads. One is the creation overhead, which is the time required to create and start a VM before it is ready to run tasks. The other one is the migration overhead, which is the one incurred when moving a running VM between two different nodes. When a new VM needs to be started in the system the time to create and boot up it in each host is considered as a cost. In the same way, the time required to migrate a VM is also taken into account for that new tentative allocation. This cost reduces the number of migrations and so, prevents the same VM from moving too often.

Furthermore, a migration penalty ($P_m$), which considers an estimation of the remaining execution time according to the user initial requirement, is used. This is done for penalizing the migration of those VMs which remaining execution time is small: they will finish soon and there is no need to migrate them.

$$
\begin{aligned}
T_c(h, vm) &= \text{time of creating } vm \text{ in } h \\
T_m(h, vm) &= \text{time of migrating } vm \text{ to } h \\
T_u(vm) &= vm \text{ execution time according to user} \\
t(vm) &= \text{time since } vm \text{ submission} \\
T_r(vm) &= vm \text{ remaining time according to user} \\
&= T_u(vm) - t(vm) \\
P_m(h, vm) &= \begin{cases} 2 \cdot T_m(h, vm) & T_r(vm) < T_m(h, vm) \\ T_m(h, vm) & T_r(vm) \geq T_m(h, vm) \end{cases}
\end{aligned}
$$

Furthermore, in order to prevent possible VM migrations or other actions while the same virtual machine is being operated (created, migrated, checkpoint...), we set an infinity penalty while any action is being performed in a given VM.

$$
C_{virt}(h, vm) = \begin{cases} 0.0 & \text{if } vm \text{ is in } h \\ \infty & \text{if action in } vm \\ T_c(h, vm) & \text{if } vm \text{ is new} \\ P_m(h, vm) & \text{otherwise} \end{cases}
$$

Another factor is the concurrency, performing more than one action at the same time can generate a race for the resources (e.g. disk, CPU) which will add an additional overhead. We calculate a concurrency penalty for each host that indicates whether it is already creating or migrating a VM. This cost is applied to those VMs which can be created or moved to that node.

$$
C_{state}(h, vm) = \begin{cases} T_c(h, vm) & \text{if } h \text{ is creating } vm \\ T_m(h, vm) & \text{if } h \text{ is migrating } vm \\ 0.0 & \text{otherwise} \end{cases}
$$

$$C_{conc}(h) = \sum_{vm \text{ in } h} C_{state}(h, vm)$$

$$C_{conc}(h, vm) = \begin{cases} 0.0 & \text{if } vm \text{ is in } h \\ C_{conc}(h) & \text{otherwise} \end{cases}$$

This virtualization cost can also take into account dependencies between VMs. For instance, if a provider want to deploy an application server and a database, this would avoid to deploy them in different providers.

These costs of operating with VMs are caused by extra time ($T_{op} = C_{virt}(h, vm) + C_{conc}(h, vm)$) and this needs to be expressed in economic currency (*time is gold*). A VM in operation time implies different costs. First, the fixed cost of maintaining the host machine. Second, the power used by the host machine while not taking profit of the VM execution. And third, the price of renting the host while not executing the VM content.

All these factors are turned into economical values using this conversion function:

$$C_{op}(h, vm) = \frac{\frac{C_{fix}(h)}{hour} + \frac{C_{pwr}(h)}{hour} + \frac{Price(VM)}{hour}}{3600} \cdot T_{op}$$

### D. Solving scheduling

Once the scheduling knows the revenue and costs of executing each VM $VM_i$ in each host $H_j$, it puts all this information together in a matrix of type $\langle$host,VM$\rangle$. This aggregation follows the next formula:

$$C(h, vm) = \{C_{req}(h, vm), C_{fix}(h), C_{var}(h, vm), C_{op}(h, vm)\}$$

$$B(h, vm) = R(vm) - \sum C_i(h, vm)$$

Using this matrix, it tries to find those combinations with better benefit for the overall system. This process firstly consists in subtracting from each cell $\langle H_j, VM_i \rangle$ the current benefit of maintaining $VM_i$ in the current host (i.e. this is the value of cell $\langle H_{cur}, VM_i \rangle$ if $VM_i$ is running in $H_{cur}$). After this, it obtains for each cell the difference (improvement or degradation) of moving a VM from its current host to the host corresponding to this cell. Positive scores mean improvement and negative scores mean degradation.

Having the matrix preprocessed, optimization process can start by selecting on each iteration the biggest value of the matrix representing the best movement to be done in all the system. After moving the corresponding VM to the new host machine, the matrix is refreshed with new scores. The main idea is to iterate until the cost matrix has no positive values. However, there is always the possibility of not converging and entering in a periodic movement cycle, so a limit number of movement per scheduling round is applied.

When the matrix reaches a state where all values are negative or zero (no improvements can be done), or the number of movements has reached a given limit, it is assumed a suboptimal solution for the current system configuration has been found. This optimization is shown in Algorithm 1.

```
M    := Matrix [hosts][VMs];

- Fill M:
    - Add revenues for each VM;
    - Substract costs for each Host,VM;

While M has positive values do:
    <h,v>  := biggest position on M;
    o      := current host for v;

    - Re-schedule VM v from Host o to Host h;
    - Recalculate M values;

    If (iterations limit reached) then:
        break;
    End If
End While
```

Algorithm 1: Algorithm for allocation matrix optimization

Note that this is an algorithm based on Hill Climbing which is greedy. Nonetheless, in this situation it finds a suboptimal solution much faster and cheaper than evaluating all possible configurations. Each step brings to a more optimal configuration until there are no better configurations or an iteration limit is reached. The complexity has an upper boundary of $O(\#Hosts \cdot \#VMs) \cdot C$ since it iterates over the $\langle$host,VM$\rangle$ matrix $C$ times. In addition, in the current study case, some of the constraints help to reduce the search space, i.e. the resource requirement constraint discards a great amount of combinations at the beginning of the algorithm.

## IV. System actuators

Once the scheduling policy has decided the host allocation for each VM, changes needs to be performed in the system using the facilities that virtualization offers, such as VM creation and migration. In case the VM has never been running in the system, the scheduler invokes the selected node to create this VM. If a given VM has been moved from a node to another, the scheduler asks the current executing node to migrate it to its new location. Furthermore, if the VM was running in a failed node, the new executing node tries to recover it from the more recent checkpoint, and if there is not available checkpoint, it recreates the VM. Next subsections present into detail some of the actuators.

### A. Virtual host

In order to implement a queue with new VMs to be created or those failed, a special host is added, namely the *virtual host*. It acts as a queue where not allocated VMs are temporary scheduled, this is got by assigning an infinite cost to those VMs held in that host, so the penalty of keeping them without real allocation is the maximum one. Hence, the operations with maximum benefit will be those involving the allocation of a new VM or failed ones into a real host which is able to handle it.

### B. Outsourcing

The option of outsourcing any VM to an external third-party IaaS provider is also considered. If the local costs are

too big compared with the profit (e.g. there is not enough local capacity to execute a VM in the local provider), the scheduler can decide to start a VM in an external provider.

In order to support this case, a special host ($h_p$), which represents an external provider, is added to the local one. We add so many special hosts as external providers we are outsourcing to. When considering an external provider, the cost model explained in Section III is simplified considerably: (1) the occupation of this host $O(h_p, vm)$ is considered zero (2) availability of resources in the other provider is taken into account when evaluating $Req(vm)$ in $h_p$ (3) possible migrations of a given $vm$ from and to those special hosts are avoided ($T_m(h, vm) = \infty$; $T_m(h_p, vm) = \infty$) and there is not any concurrency penalty ($C_{conc}(h_p, vm)$ = 0), so $C_{virt}(h_p, vm)$ is equal to the cost of creating a VM in the external host *h* ($T_c(h_p, vm)$); (4) $C_{pwr}(h_p, vm)$ is equal to zero because executing a VM into an external host does not incur any energy cost for the provider; and (5) there is no fixed costs, so the execution cost is equal to $C_{var}(h_p, vm)$, which in this case is determined by the cost of renting the VM into an external host $C_{rent}(h_p, vm)$.

### C. Fault tolerance

The proposed model is also able to support fault tolerance. Each host has a given availability factor ($Up(h)$) between 0 and 1 which will be zero if it is 90% or less (this is set as minimum uptime) and 1 if the node is always up, i.e. there are no failures. In order to take this into account, we multiply the final benefit of an allocation by this uptime factor, which can be expressed as $B(h, vm) = Up(h, vm) \cdot B'(h, vm)$. However, some VMs have permissiveness to failure, so no uptime factor is applied to them.

$$\begin{aligned} F_{tol}(vm) &= \text{if } vm \text{ tolerate failures} \\ Up(h, vm) &= \begin{cases} 1 & \text{if } F_{tol}(vm) \\ Up(h) & \text{otherwise} \end{cases} \end{aligned}$$

In addition, our approach supports the capability to recover executions from a checkpoint. In this sense, the system periodically decides to perform a checkpoint according to the profitability of doing this checkpoint. First of all, it checks if the VM is stateless and in that case, it does not perform any checkpoint since it is not useful. Otherwise, it checks the execution time elapsed after the last checkpoint and if it is worth taking into account the required time to perform a checkpoint, it will perform the checkpoint process.

Finally, in order to complete the recovery mechanism, when a node fails, the VMs that were being executed on that host are moved to the *virtual host* with an infinity cost. Once a checkpointed VM has been scheduled to be executed again, it is recovered from the last checkpoint.

### D. Power on/off

One of the key decisions is determining the amount of operative nodes or in other words, when a node can be turned off in order to save power consumption, or turned on again in order to be used to fulfill the tasks SLAs in the same way it is done in [14]. This decision is driven by the cost of maintaining a node and two thresholds: the minimum *Working hosts* threshold $\lambda_{min}$ and the maximum *Working hosts* threshold $\lambda_{max}$. When the ratio of working nodes ($fracnumber_of_working_nodesnumber_of_operative_nodes$) goes over $\lambda_{max}$, the scheduler must start turning on stopped nodes. The nodes to be turned on are selected according to a number of parameters, including its reliability, boot time, etc. On the other hand, when the ratio of working nodes goes below $\lambda_{min}$, the scheduler can start turning nodes off. The scheduler selects those idle machines according to their cost. This cost results from the aggregation of benefits (matrix row) and taking into account the number of negative infinities. Those nodes with a lower global benefit are selected to be turned off. Finally, in order to define a minimum set of operative machines, the scheduler can use the $\min_{exec}$ parameter.

## V. Experimental environment

### A. Simulator

The presented scheduling will be tested on top of a virtualized data center which executes batch jobs and hosts web services in the same way it was done in [15]. As it is difficult to work in this kind of environments, a platform which simulates this kind of provider behavior has been built. This takes into account the virtualization overheads (including creation, migration, checkpoint), the ability to turn nodes on and off (including the ability to simulate node crashes), and the power consumption. An early stage of this simulator was already presented in [14], where a framework for achieving energy-efficient data centers by using machine learning techniques and power-aware consolidation algorithms was presented.

In order to model the performance of the applications that will be executed, we have used different CPU intensive tasks (with several CPU consumptions) to model HPC jobs, and the SPECweb2009 e-Commerce application [16], to model Web-based applications. This model has been obtained by stressing this application (deployed on a Tomcat v5.5 with an hybrid architecture) with different input loads and with different processing units. Indeed, this modeling has been focused on the response time high-level metric and, in this sense, it has been detected a performance pattern which relates this metric with both incoming users' load and CPU usage of the server (note that the application used is CPU-bounded in this environment). The aforesaid pattern has been divided in three phases in order to simulate the server's performance: an *stationary* response time when the incoming load causes a CPU utilization less than 60%; *slightly increased* when this CPU utilization is between 60% and 80%; and *pronounced linear* when the server is overloaded (i.e. CPU utilization greater than 80%).

In order to validate the simulator power consumption, a real workload has been submitted to a single node which provokes different situations and we have measured its CPU usage and power consumption. This validation process shows our simulator has an error of less than -0.43 Wh over 93.49 Wh of real power consumption. Regarding the instantaneous error, it has less than 6.23 W of absolute error which represents relative error of 0.02%. Figure 1 shows this validation by comparing the measured consumption and the simulated one.

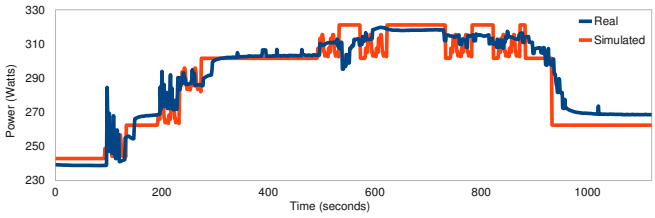


Figure 1. Simulator power consumption validation.

### B. Data center workload

Our approach is able to deal with an heterogeneous workload composed by batch HPC jobs and Web-based services. For testing purposes, the former is a Grid workload obtained from Grid5000 [17] on the week that starts on Monday first of October of 2007. The latter workload was obtained from an anonymous European ISP and was collected during 2009. It results from the aggregation of several services and the used profile is of a whole week, thus representing different classical increases of load of each weekday and the decreases of load of weekends.

### C. Service Level Agreements

This provider offers a SLA to its users which specifies the price they have to pay for executing a given amount of time of any task on the provider, in a similar way Amazon EC2 [18] does. The SLA also defines penalties according to the type of application. On the one hand, HPC jobs have a soft deadline where starts to penalize and a hard deadline where it reaches the maximum penalty. On the other hand, Web-based applications and services have a SLA composed by different IT metrics, such as the *availability*, expressed as an uptime percentage, and the *performance*, determined by the response time (in seconds). Actually, using the model presented in Section III, it is performed a linkage between this IT (low-level) metrics with business (high-level) metrics, such as the benefits, revenue, costs, etc.

Once the provider knows the percentage of time in which each SLA has been violated $\sigma(S_i)$, it uses this violation ratio as the input for determining the SLA penalty $Pen(S_i)$. More about, these penalties are calculated as a percentage of the revenue acquired $Rev(S_i)$ when fulfilling the corresponding SLA $S_i$: $Pen(S_i) = Rev(S_i) \cdot \frac{Gom(\sigma(S_i))}{100}$. In fact, it is measured by using a Gompertz function $Gom(\sigma(S_i))$, which is a kind of sigmoid function where growth is slowest at the start and end of a time period. Considering those premises, SLA is used as a metric of the system's performance.

### D. Scheduling policy configuration

The experiments consist of the simulation of a whole virtualized data center with 65 nodes. The data center is configured to have three different types of nodes according to their virtualization overheads. There are 10 fast nodes, with $T_c$ = 30s and $T_m$ = 40s, 35 medium nodes with $T_c$ = 40s and $T_m$ = 60s, and 20 slow nodes with $T_c$ = 60s and $C_m$ = 80s.

Different revenues and costs are setup in order to simulate a real provider. Regarding the revenues, medium EC2 instances with high CPU load are assumed, which have a cost of 0.17 €/hour (EC2 pricing in Europe). As an external provider, we consider two Amazon EC2 data centers: US and Europe. Both have an average creation time of 300 seconds and a cost of 0.19 and 0.17 respectively.

The electricity pricing used is the Spanish one, that is 0.09 €/KWh [19]. Furthermore, in order to calculate the cost of the nodes, we also take into account the amortization of the servers (in 4 years) and the space (in 10 years) required to deploy them using a price of 1000 $€/node$ and 2000 $€/m^2$, respectively. Assuming 10 nodes per $/m^2$, it implies 0.03 €/h per each node in the data center.

## VI. Evaluation

This section evaluates the proposed scheduling policy comparing it against common scheduling techniques when executing the already introduced heterogeneous workload. This evaluation takes into account different parameters such as power consumption (Pwr), number of nodes that are working (Work) and turned on (Run), Quality of Service (QoS), and final provider's benefit (B). In order to measure the QoS, the following metrics for defining the SLA penalties are used: HPC jobs use the deadline ($D$), and Web-applications use the response time $R$. Both expressed in percentage where 0% is always violating and 100% the SLA is always fulfilled.

### A. Virtualization overheads

This section evaluates the effect of taking care of virtualization overheads. Table I presents the results of static scheduling, which just assigns a node when the VM is submitted and does not move it. The static policies are: *Random* (RD), which assigns the tasks randomly; *Round Robin* (RR), which assigns a task to each available node and, therefore, this implies a maximization of the amount of resources to a task but also a sparse usage of those resources. We also evaluate a common *Backfilling* (BF) policy, which tries to fill as much as possible the nodes, thus solving the former problem; and different versions of our policy. The first one (E0) just takes into account hardware, software

and resource requirements $C_{req}$, power efficiency $P_{pwr}$, and does not perform migrations. Results show RD and RR get very bad performance, and BF and E0 get a very similar one. E1 policy, which extends E0 by taking into account the virtualization overheads, allows the provider getting a better benefit by reducing the SLA penalties.

Table I
SCHEDULING RESULTS OF POLICIES WITHOUT MIGRATION

| | Work/Run | Pwr (kW) | $D$ (%) | $R$ (%) | B (€) |
|---|---|---|---|---|---|
| RD | 19.6 / 28.8 | 1693.7 | 70.3 | 79.2 | 272.0 |
| RR | 21.3 / 28.4 | 1388.6 | 69.6 | 83.2 | 448.4 |
| BF | 16.3 / 32.3 | 1560.5 | 95.2 | 84.3 | 503.6 |
| E0 | 16.2 / 32.8 | 1569.1 | 95.8 | 84.3 | 503.0 |
| E1 | 16.2 / 32.9 | 1584.5 | 96.9 | 84.4 | 509.0 |

Table II
SCHEDULING RESULTS OF POLICIES USING MIGRATION

| | Work/Run | Pwr (kW) | $D$ (%) | $R$ (%) | Mig | B (€) |
|---|---|---|---|---|---|---|
| DBF | 15.8 / 31.1 | 1499.6 | 95.74 | 84.30 | 114 | 494.4 |
| E2 | 15.1 / 29.4 | 1435.1 | 97.20 | 84.21 | 1170 | 533.1 |

Table II shows the results of scheduling policies that use migration capabilities in order to improve consolidation: *Dynamic Backfilling* (DBF), which applies *Backfilling* and migrates VMs between nodes, and our policy E2, which considers all the costs and includes the migration capability. While DBF is not enough since it does not take into account the introduced virtualization overheads and tries to make a conservative consolidation (114 migrations), E2 is able to increase the benefit of the provider by performing an extreme consolidation (1082 migrations) as it focuses on economic parameters and manages virtualization.

### B. Fault tolerance

This section demonstrates the ability of the presented scheduling to deal with crashes. The experiment simulates a real environment where nodes crash with a given probability. In particular, the nodes crash one or two times during the test week in average, getting a 99.98% and 99.99% uptime respectively.

Table III
SCHEDULING RESULTS OF POLICIES WITH A FAULTY ENVIRONMENT

| | Work/Run | Pwr (kW) | $D$ (%) | $R$ (%) | B (€) |
|---|---|---|---|---|---|
| DBF | 14.7 / 20.4 | 537.1 | 95.6 | 91.7 | 311.3 |
| E2 | 19.7 / 32.0 | 1338.4 | 95.4 | 86.1 | 495.8 |
| EFT | 18.4 / 29.7 | 1232.7 | 95.4 | 85.1 | 518.2 |

Table III compares the behavior of DBF, the simple economic policy (E2), and an extended version of this policy which takes into account fault tolerance and performs checkpoint management (EFT). This experiment demonstrates DBF is not able to deal with failures and loses many applications which makes it losing their the revenue. E2 is able to resubmit failed applications, but it gets low benefit as SLA fulfillment is compromised. Finally, EFT is able to recover long executions from a previous checkpoint and gets better SLA fulfillment. Thanks to this policy, provider can improve its benefits in a faulty environment.

### C. Outsourcing

Previous results has big fixed costs of more than 350€for maintaining a data center with 65 nodes during a week, while not all of them are used all the time. In order to tackle this high cost, the presented policy takes advantage of outsourcing technique in order to withstand periods of high load. This experiment consists on reducing the amount of nodes to 30 and add two different external providers.

Table IV
SCHEDULING RESULTS OF POLICIES INTRODUCING OUTSOURCING

| | Nodes | Work/Run | Pwr(kW) | $D$(%) | $R$(%) | Out | B(€) |
|---|---|---|---|---|---|---|---|
| E2 | 65 | 15.1 / 29.8 | 1454.6 | 97.2 | 84.21 | 0 | 529.1 |
| EO | 65 | 14.3 / 27.7 | 1356.1 | 97.0 | 84.6 | 15 | 450.9 |
| E2 | 30 | 16.2 / 26.0 | 1280.9 | 64.2 | 84.2 | 0 | 429.7 |
| EO | 30 | 14.9 / 25.3 | 1236.7 | 96.1 | 84.9 | 477 | 616.3 |

Table IV shows the difference between using outsourcing (EO) or not when having enough resources for the whole workload (65 nodes) is not very big. In addition, when the provider has only 30 nodes, it has bigger SLA penalties and reduces the benefit of the provider. Nevertheless, it gets a bigger benefit when using outsourcing as it is able to reduce the amount of local resources and rent peak loads to external resources which increases the provider benefit up to 15%.

## VII. CONCLUSIONS

In this paper, we have presented a scheduling policy that takes advantage of virtualization in order to consolidate multiple heterogeneous workloads and save power by turning off idle nodes. Nevertheless, managing a data center which supports virtualization implies dealing with many other factors. In order to simplify this problem, considering revenues and all the costs related with the execution of a virtual machine has demonstrated to be a powerful way. According to this, our policy is able to take into consideration different costs like hardware pricing and its amortization, power consumption, virtualization overheads such as migration or creation, and SLA penalties due to low performance.

This methodology is generic enough to include any other cost like for outsourcing, which would also include bigger overheads with a higher pricing. This exemplifies how the presented formulation of the problem is intuitive, captures the most important considerations in managing a virtualized data center, and lends itself easily to extension.

Other challenges like fault tolerance have also been taken into account, as our system is able to recover part of the previous task execution and reduce the amount of computing time required in a faulty environment, making it more efficient and reliable. We have also demonstrate how this approach is able to aggregate so different targets like fault tolerance and power efficiency.

In addition, the results obtained in this paper show a benefit of more than 15% with respect to typical policies, which is closer to a maximization of the provider's benefit. The experiments using a heterogeneous real workload, composed by a real Grid workload and a Web service-based one, demonstrates how the policy deals well with virtualization costs. In addition, experiments exemplify that these techniques can offer substantial improvements in energy and performance efficiency in these scenarios.

Our future work will focus on extending the proposed policy by evaluating parameters such as dynamic SLA enforcement or adding more heterogeneity support. Moreover, new enhancements to the scheduling policy like dynamic thresholds will be studied, as well as, the improvement of the accuracy of current modeling.

## ACKNOWLEDGMENT

This work is supported by the Ministry of Science and Technology of Spain under contract AP2008-0264, TIN2008-06582-C03 and TIN2008-06582-C03-01, by the Generalitat de Catalunya under contracts 2009-SGR-980 and 2009-SGR-1428, and the European Union under contract TIN2007-60625, and EU PASCAL2 Network of Excellence (FP7-ICT-216886).

## REFERENCES

[1] R. Nathuji, K. Schwan, A. Somani, and Y. Joshi, "VPM tokens: virtual machine-aware power budgeting in datacenters," *Cluster Computing*, vol. 12, no. 2, pp. 189–203, 2009.

[2] V. Petrucci, O. Loques, and D. Mossé, "A framework for dynamic adaptation of power-aware server clusters," in *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*. New York, NY, USA: ACM, 2009, pp. 1034–1039.

[3] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of Job-scheduling Strategies for Grid Computing," *1st IEEE/ACM International Workshop on Grid Computing, Bangalore, India, December 17, 2000*, pp. 191–202, 2000.

[4] L. Liu, H. Wang, X. Liu, X. Jin, W. He, Q. Wang, and Y. Chen, "GreenCloud: a New Architecture for Green Data Center," in *6th International Conference on Autonomic Computing and Communications, Industry Session, Barcelona, Spain, June 15–19, 2009*. ACM, 2009, pp. 29–38.

[5] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, no. 1, pp. 20–26, 2008.

[6] R. Bianchini and R. Rajaniony, "Power and Energy Management for Server Systems," *IEEE Computer, Special issue on Internet data centers*, vol. 37, no. 11, pp. 68–76, 2004.

[7] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing Server Energy and Operational Costs in Hosting Centers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 303–314, 2005.

[8] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 103–116, 2001.

[9] B. Chun, G. Iannaccone, G. Iannaccone, R. Katz, L. Gunho, and L. Niccolini, "An Energy Case for Hybrid Datacenters," *Workshop on Power Aware Computing and Systems (HotPower'09), October 10, Big Sky, MT, USA*, 2009.

[10] Y. Chen, D. Gmach, C. Hyser, Z. Wang, C. Bash, C. Hoover, and S. Singhal, "Integrated Management of Application Performance, Power and Cooling in Data Centers," *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)*, 2010.

[11] S. Fu, "Failure-Aware Construction and Reconfiguration of Distributed Virtual Machines for High Availability Computing," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 372–379.

[12] K. Le, R. Bianchini, M. Martonosi, and T. Nguyen, "Cost- and Energy-Aware Load Distribution Across Data Centers," *Workshop on Power Aware Computing and Systems (HotPower'09), October 10, Big Sky, MT, USA*, 2009.

[13] Í. Goiri, J. Guitart, and J. Torres, "Characterizing Cloud Federation for Enhancing Providers' Profit," in *Proceedings of the 3rd International conference on Cloud Computing (CLOUD 2010), Miami, Florida, USA, July 5-10*, 2010, pp. 123–130.

[14] J. Berral, Í. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavalda, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning," in *1st International Conference on Energy-Efficient Computing and Networking (eEnergy'10), University of Passau, Germany, April 13-15*, 2010, pp. 215–224.

[15] J. O. Fitó, Í. Goiri, and J. Guitart, "SLA-driven Elastic Cloud Hosting Provider," in *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP'10), Pisa, Italy, February 17-19*, 2010, pp. 111–118.

[16] "SPECweb2009 E-commerce workload," 2009, http://www.spec.org/web2009/docs/design/EcommerceDesign.html.

[17] "The Grid Workloads Archive," 2009, http://gwa.ewi.tudelft.nl.

[18] "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2.

[19] "Europe's energy portal," http://www.energy.eu.